Secure Data Storage on the Cloud using Homomorphic Encryption

Manoj Kumar Mohanty

Department of Computer Science and Engineering
National Institute of Technology Rourkela
Rourkela – 769 008, India

Secure Data Storage on the Cloud using Homomorphic Encryption

Thesis submitted in

June 2013

to the department of

Computer Science and Engineering

of

National Institute of Technology Rourkela

in partial fulfillment of the requirements

for the degree of

Master of Technology

in

Computer Science and Engineering

(Specialization: Information Security)

by

Manoj Kumar Mohanty

(Roll 211CS2282)

under the supervision of

Dr. Ashok Kumar Turuk

Department of Computer Science and Engineering

National Institute of Technology Rourkela

Rourkela – 769 008, India

Computer Science and Engineering
National Institute of Technology Rourkela
Rourkela-769 008, India. www.nitrkl.ac.in

Dr. Ashok Kumar Turuk
Associate Professor

June 4, 2013

Certificate

This is to certify that the work in the thesis entitled ***Secure Data Storage on the Cloud using Homomorphic Encryption*** by ***Manoj Kumar Mohanty***, bearing roll number 211CS2282, is a record of an original research work carried out by him under my supervision and guidance in partial fulfillment of the requirements for the award of the degree of *Master of Technology* in *Computer Science and Engineering* with specialization in *Information Security*. Neither this thesis nor any part of it has been submitted for any degree or academic award elsewhere.

Ashok Kumar Turuk

Acknowledgment

I would like to take this opportunity to thank each and every person involved in the successful completion of this thesis. First of all I thank my thesis supervisor Prof. Ashok Kumar Turuk for the constant support and encouragement that he has provided me throughout the course of this thesis. I would like to express my gratitude to him for the guidance and valuable advice that he has provided, and for being a constant source of motivation.

I thank all the professors of the department of Computer Science and Engineering for the advices, resources and environment they have provided for the successful completion of my work. The thesis would not have been successful without their support. Besides, I thank my friends and peers who have been a source of inspiration for the work.

I must acknowledge the academic resources that I got from National Institute of technology Rourkela. I would like to thank the administrative and technical staff members of the Department who have been kind enough to advise and help in their respective roles.

I would also like to thank my parents and family members for their love and support, which has been a guiding force for the work I have done.

Manoj Kumar Mohanty

Abstract

Organizations are showing great interest in storing data on public clouds. This could be a result of the unprecedented growth of data recorded in the last few years. However the security issues associated with data storage over cloud is a major discouraging factor for potential adopters. Hence the focus of today is to find cryptographic techniques that will offer more than confidentiality. Homomorphic encryption is one such method that has interesting applications in cloud. The objective is to manage and protect the data from the users of a client organization which wants to store the data on untrusted public clouds. In this thesis a hybrid cloud framework is proposed that addresses the privacy and trust issues and provides encrypted storage with public clouds. The proposed method uses Homomorphic Encryption for protecting the user data and uses a modified file updation technique to reduce bandwidth consumption during transfer of large encrypted files.

***Keywords*:** Cloud Computing, Cryptography, Cloud Storage Security, Fully Homomorphic Encryption, Delta Encoding.

Contents

Certificate ii

Acknowledgment iii

Abstract iv

List of Figures vii

List of Tables viii

1 Introduction 1

1.1 Motivation 2

1.2 Objectives 3

1.3 Organization of Thesis 3

2 Cloud Computing: A Security Perspective 5

3 Homomorphic Encryption 13

3.1 Mathematical Background 15

3.1.1 Integer Factorization Problem 15

3.1.2 Discrete Logarithm Problem 16

3.1.3 Quadratic Residuosity Problem 16

3.2 Homomorphic Cryptosystems 16

3.2.1 RSA Cryptosystem 16

3.2.2 Paillier Cryptosystem 18

3.2.3 Boneh-Goh-Nissim Cryptosystem 19

4 Secure data storage using Homomorphic Encryption **22**

4.1 Current Scenario 23

4.2 Proposed Framework 23

4.2.1 Components 25

4.2.2 Updation Methods 28

5 Implementation and Results **33**

5.1 Experiment Details 34

5.2 Performance Analysis 34

6 Conclusion and Future work **38**

Bibliography **40**

List of Figures

2.1 Cloud Computing Overview 6

4.1 Framework Usage Scenario 24
4.2 Overview of Framework Components 25
4.3 File information on FILES database. 28

5.1 Variation of patch file sizes over instances 36
5.2 Variation of α over instances 37

List of Tables

5.1 Various file size information for 3 instances. 35

Chapter 1

Introduction

Motivation

Objectives

Organization of Thesis

Cloud computing offers a cost-effective solution to manage the IT infrastructure in a flexible and scalable manner. Cloud computing enables software applications, deployment platforms, even the computing resources to be made available on-demand using a pay-as-you-go model. This has drawn a lot of attention towards the domain in recent years. Today a good number of organizations use the cloud for their day to day operations and the adoption rate by others are also high [1].

Hosted applications over the Internet have evolved greatly. The web which originally just consisted of static web pages, today serves as platform for many web applications that ranges from simple note taking tools to computation intensive scientific simulation services. One thing that makes such an approach special that users can outsource data and computation to a remote server that has enough resources to perform the task within much less time than traditionally running an equivalent application on the user's machine. This is also one of the major factors that are driving the research in the cloud computing technologies.

However there have been concerns with respect to confidentiality and privacy of data being stored on the clouds. Specifically organizations that handle sensitive and high-risk data such as medication records, financial details are not convinced with the security measures currently in-place to protect the data. Further many public clouds fail to meet the regulatory guidelines required for operations of such applications.

In recent times there have been reports of many security breaches of cloud services such as Dropbox [2,3], Last.fm [4,5], and iCloud [6,7]. A study [8] suggests 72% of the IT professionals blame employees for most data breaches, whereas the rest blame the hackers. It also reveals that 32% data was lost while 18% data was stolen by employees. This increases the concern of insider attack on public clouds.

1.1 Motivation

In a recent survey [1], 79 % employees said their organization uses SaaS and 45 % agreed on using IaaS. These numbers indicate a considerable growth and interest in cloud adoption. In time to come more and more organizations are going to adopt

cloud based solutions because of the benefits of the cloud infrastructure over the local traditional IT infrastructure. This will result in moving a large amount of data to the cloud. Hence more investigation into cloud storage frameworks is needed so as to make the cloud adoption simple and user friendly. But as the rate of cloud adoption is growing, the security risks associated with the networked applications in general and cloud in specific is also growing. This is a major discouraging factor towards large scale cloud adoption. Due to the security risks, organizations that are dealing with sensitive information have ignored the cloud in the past. But as more amount of data is being generated, consumed, processes and stored in digital form, the operational cost of a traditional IT infrastructure is becoming very high. Hence organizations that deals with sensitive information such as health care records are also considering cloud as an option. Hence the need of secure cloud storage options are growing.

1.2 Objectives

The objective is to develop a framework using which an organization can store its data on the cloud in a secure manner. The requirements for the framework are as follows. It should be easy to use and should not depend upon security measures taken by the end users e.g. the employees of the organization or users of a service provided by the organization. It should handle all the cryptographic operations within the trusted infrastructure of the organization and then send the encrypted data to the cloud. The public clouds in which the encrypted data is stored should not have the ability to decrypt the content. It should handle file uploads in an efficient manner to reduce bandwidth consumption.

1.3 Organization of Thesis

Rest of the thesis is organized as follows. Chapter 2 describes cloud computing from a security perspective. It includes notes on recent works on security assessment of cloud. Chapter 3 describes Homomorphic Encryption schemes. It

provides details on the homomorphic properties of some cryptosystems and notes on recent developments in this area. In Chapter 4 a framework for secure data storage on cloud is proposed. A detailed description of the framework along with all components are described in the chapter. Chapter 5 provides notes on the implementation of the framework introduced in Chapter 4 and it also describes the performance of the framework. Chapter 6 provides concluding remarks and notes for possible enhancements.

Chapter 2

Cloud Computing: A Security Perspective

In simple words, Cloud computing can be described as a method that allow resources to be made available over a network in general and Internet in specific. A more formal definition by NIST defines cloud computing as follows. Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction [9]. NIST definition also includes five essential characteristics, three service models, and four deployment models for cloud. An overview of the same is presented in Figure 2.1.

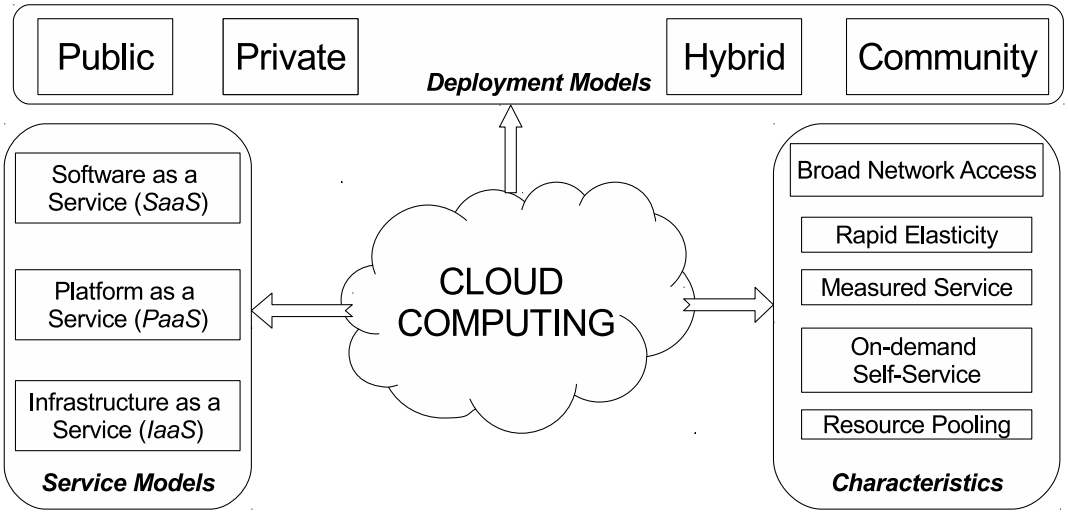

Figure 2.1: Cloud Computing Overview

Essential Characteristics of the cloud:

On-demand self-service: It allows a consumer to provision resources such as server, storage, network and the like as required without manual interaction with service providers. This facilitates quick deployment and provides the consumers flexibility to make any changes as they see fit.

Broad network access: The services offered by the cloud are provided over the network using standard protocols so that the services can be easily accessed from a number of client devices such as workstations, laptops, tablets, and mobile phones.

This gives consumers the flexibility to choose a platform of their choice to work with.

Resource Pooling: The cloud follows a multi-tenant model in which computing resources are pooled and the resources are dynamically allocated as per the need of the consumers. This provides location independence as the consumer is not aware of the exact physical location of a resource in the cloud. Although the physical location from where the resource should reside can be controlled at a higher abstraction level by specifying certain geographical limits. As an example a consumer in some country can specify to keep its storage limited within physical infrastructure of the home country only.

Rapid Elasticity: Additional resources can be easily provisioned or released as per the demand. In the cloud the resource availability appear virtually unlimited and resources can be requested in any quantity depending upon requirements. The cloud should handle the scaling as needed.

Measured Service: In order to maintain a transparent record of the resource usage both for the providers and the consumers the cloud systems often use a metering capability depending on the kind of service being provided. This helps many activities such as billing, setting quotas on resource usage, and auditing.

Service Models in Cloud Computing:

Software as a Service (SaaS): It includes services where the consumers are given access to applications deployed on the cloud infrastructure of the providers. The applications are accessed using various device platforms through web browser, or some native applications interface. In this case the consumers do not have the control over the cloud infrastructure. Still they can specify certain limited configuration settings.

Platform as a Service (PaaS): It provides the consumer the ability to deploy an application onto the cloud infrastructure of the provider using development environment supported by the provider. Here the consumers do not have the control over the cloud infrastructure of the provider but do have good control over the applications deployed by them.

Infrastructure as a Service (IaaS): It provides the consumers ability to provision

basic computing resources such as servers, storage, and networks on top of the cloud infrastructure of the provider. In this case the consumers have more control over the resources provisioned and can configure it as needed e.g. the consumers have choice in selecting operating system or installing any software on a provisioned virtual machine. But the consumers still do not manage or control the cloud infrastructure of the provider.

Deployment Models in Cloud Computing:

Private Cloud: This type of cloud infrastructure is provisioned by a single organization for its exclusive use. It allows the organization more control over the underlying infrastructure and gives feeling of enhanced security. It may be managed by by the organization or some third party.

Community Cloud: This type of cloud infrastructure is targeted towards a selected group or community of consumers belonging to organizations with related goals. It may be managed by one or more of the organizations involved, or by some third party.

Public Cloud: This kind of cloud infrastructure is available for use by the public. Public cloud services are getting a lot of attention as it reduces the efforts needed for managing the infrastructure. This allows the consumers to focus more on their objectives. However it also raises many security concerns. It is mostly managed by the providers themselves.

Hybrid Cloud: This kind of cloud infrastructure comprises of two or more forms of the previous cloud infrastructures where interactions among the different infrastructures are made possible by using standardized or proprietary data and application interaction methods.

Recent works done in the context of cloud security is briefly described below. Chen and Zhao provides an analysis [10] of data security throughout seven phased data life cycle namely Generation, use, transfer, share, storage, archival, and destruction. In data storage, they have mentioned key management as an important point. Again in data destruction phase it is important to ensure that data is securely erased using methods that makes it unrecoverable. Otherwise an adversary can take advantage of the physical characteristics of the storage medium

to access sensitive information.

[11] addresses security issues related to both single and multi-cloud models. Authors have evaluated the security of single clouds based on three factors, namely data integrity, data intrusion, and service availability. Data integrity is important since the data can get corrupted during transmission or transfer between data source and the cloud. The scenarios involving use of multiple clients and devices for interactions with the cloud further complicates the data integrity requirement. Another security factor is the data intrusion, in which the adversary gains access to a cloud service through stolen passwords and then can cause damage to the services being used by the genuine users. Service availability is also another factor to be considered.

An execution model called HybrEx (Hybrid Execution) model has been proposed by Ko, Jeon, and Morales [12]. This model is designed with a view to ensure confidentiality and privacy. It operates by utilizing the public clouds for non-sensitive data and computation, while the sensitive data and computations are performed with in the private cloud infrastructure of the organization. The model supports application level partitioning i.e. when an application works with both public and private data, two separate partitions of the applications are made, one that involves the public data is made to run on the public cloud whereas the other one runs on the private cloud. It also allows integrations of additional computing or storage resources to the private cloud from public clouds without compromising the confidentiality or privacy of data. Although this approach provides security benefits but this model avoids the public clouds for sensitive data and computation.

Another scheme due to Jajodia, Litwin, and Schwarz utilizes client-side encryption for ensuring privacy of outsourced data [13]. This scheme involves use of symmetric keys to protect data and uses Diffie-Hellman scheme for authenticating clients. Here the data is encrypted using a symmetric encryption with a unique key. The keys are cached at the client side and are also kept on the cloud as backup. A key on the cloud is kept hidden in a Public Share produced by the owner which belongs to a two-share secret. The secret is the key and the other

share called the Client Share is specific to the owner and each selected reader.

In [14] authors provide security assessment of cloud in 4 categories, namely Traditional security in a computer network, Availability of cloud computing applications, Third-party data privacy, and Third-party data control. The traditional security covers issues such as attacks on virtual machines, cloud provider weaknesses, authentication and authorization, and data stealing or leakage.

The vulnerabilities involving critical applications are described in availability of cloud computing applications. This includes cloud application uptime, single point of failure, valid computation. The third-party data privacy describes issues in context of auditability, SLAs, and design of cloud infrastructure. The third-party data control presents issues related to third-party data usage and the level of control the owner have to control it. This involves fixed response time and data deletion assurance, cloud data stealing, and losing data access.

In guidelines on security and privacy in cloud computing [15], NIST cites insider security threats as a problem against trust. Security concerns in context of data isolation in multi-tenant environment of cloud, and data sanitization measures are also described.

Another survey [16] by Subashini and Kavitha, highlights the security issues across the different service models of the cloud computing. Some of security aspects with focus on data being stored in the cloud is presented below.

Data confidentiality:

The common solution for data confidentiality is data encryption. In order to ensure the effective encryption, the key strength and the encryption algorithm both should be analyzed. As the cloud computing environment involves large amounts of data transmission, and storage hence the processing speed and computational efficiency of encrypting large amounts of data should be taken into consideration. Since large number of users are also involved hence key management is a key problem as well. Ideally the data owners should be responsible for key management. But due to the lack of expertise to manage the keys, users usually entrust the key management to the cloud providers. As the cloud providers need

to maintain keys for a large number of users, key management is getting more complex and difficult.

Data locality:

In the SaaS model of cloud computing infrastructure, consumers use the softwares and tools provided by SaaS vendors, to process their business data. The customer anyhow does not know where the data is being stored and processed. Sometimes this is not desirable because of several privacy laws prevalent in many countries. So the locality of data is very important in many enterprise applications. In many European countries, certain confidential data should not leave the country. A secure SaaS model should thus provide reliability to the customer on the location of the data.

Data integrity:

In a standalone system with a single database, data integrity is easily achieved using the database constraints. Transactional data integrity is ensured by following the ACID(Atomicity, Consistency, Isolation, Durability) properties. However in a distributed system, multiple databases and applications are maintained. Also transactions are performed along multiple data sources. To perform them in a failsafe manner, a centralized transaction management is required. SaaS applications are generally multi-tenant applications hosted by a third party. SaaS applications generally use XML based APIs. For a web service transaction management poses a problem due to lack of support for transactions at the HTTP protocol level. So API level support is required. Most SaaS vendors expose their web services APIs without any support for transactions. The lack of integrity controls at the data level can result into data corruption, hence developers must ensure that the integrity of the data is not compromised.

Data access:

The issue of data access is related to the security policies followed by the cloud provider for accessing the data. Security policies should be designed with a view to control the access to the data by the category of the user. For example regular users should be allowed to access critical or sensitive data and it should be ensured that the data is only accessible by a privileged user. Following the security policies

is the key to prevent intrusions by unauthorized users.

Data segregation:

Because of the multi-tenancy feature of the cloud infrastructure, data of several users share same physical storage location, thus giving rise to the possibility of data intrusion. This can be achieved by hacking through some vulnerability or by injecting client code. It will allow intrusion into others data. A SaaS model should thus maintain proper isolation between the data of from different users.

Data deletion: Local data deletion is often received with less attention but in case of cloud since the data is being stored in remote servers to which the user do not have physical access, hence data deletion in case of cloud is also important. A delete request for a particular data should be processed with secure data deletion practices to ensure that the data cannot be recovered later. The cloud provider should guarantee data deletion as more and more organizations are storing sensitive data in the cloud. If data is not deleted securely then it can be recovered by an adversary and will be misused.

Chapter 3

Homomorphic Encryption

Mathematical Background

Homomorphic Cryptosystems

Homomorphic encryption is a form of encryption which allows specific types of computations to be carried out on ciphertext and obtain an encrypted result which when decrypted gives the result of operations performed on the plaintext. For example, one could add two encrypted numbers and then another could decrypt the result, without either of them being able to find the value of the individual numbers.

Methods that would allow operation on data without knowing the actual content can help in lot of areas. Homomorphic encryption is one such method. Today most systems operate with help of a trusted party. Users have to trust an entity, human or machine to maintain secrecy of their data. But an attack on the trusted party or vulnerability with the system can expose the users secret. Hence the necessity of systems where even the service providers have no detailed knowledge of the users data is growing. In the next section, some of the work done in the area of homomorphic encryption is described.

Homomorphic encryption originated from the concept of privacy homomorphism [17], introduced by the Rivest et al. In their paper, they discussed about performing operations on the encrypted data. The RSA [18] cryptosystem introduced by them also exhibited the property of partially homomorphic encryption, allowing multiplication of encrypted data, which when decrypted will give the product of the plaintexts. Ever since many schemes with homomorphic properties have been proposed. Another cryptosystem developed during the same period was the ElGamal Cryptosystem [19]. Developed and named after Taher El Gamal, ElGamal cryptosystem also had some homomorphic properties. Although these two cryptosystems in their basic form is not that popular but still serves as a great introduction to the concept and many variation of the basic scheme are used in some applications. Paillier cryptosystem [20], developed by Pascal Paillier is one of the popular cryptosystem supporting additive homomorphism. Although the scheme is partially homomorphic but its simplicity and performance makes it one of the best homomorphic scheme today.

However, a cryptosystem that supports both additive and multiplicative homomorphism has been investigated since a long time. In that context,

Boneh-Goh-Nissim Cryptosystem [21] provided promising potential for a long time. It allowed unlimited addition along with one multiplication operation. But the first fully homomorphic scheme was possible due to Craig Gentry. Gentry's scheme [22] involves creating a somewhat homomorphic scheme and then bootstrapping it make it fully homomorphic.

Although homomorphic encryption has existed since a long time, but due to its computational and storage overhead it has received less attention. Further the possibility of a working fully homomorphic encryption in real world was one of the question. However in recent years development of fully homomorphic encryption schemes [23–25] have attracted a lot of focus into this field of cryptography. Homomorphic encryption has great potential for use in scenarios ranging from multi-party communication to secure computation in cloud systems.

3.1 Mathematical Background

Cryptosystems are designed in a manner such that breaking the system would require solving a problem that is intractable in nature. Some of the intractable problems that form the basis for many cryptosystems are described below.

3.1.1 Integer Factorization Problem

Integer factorization or prime factorization is the decomposition of a composite number into smaller non-trivial divisors, which when multiplied together equal the original integer. Consider two unknown distinct primes p and q and $n = p * q$. Given the product n, the difficulty in determining the primes p and q, is referred to as the factorization problem.

The general number field sieve (GNFS) algorithm is the best published algorithm for factoring large n. However in context of quantum computers, Peter Shor discovered an algorithm in 1994 that solves the factorization problem in polynomial

time. Shor's algorithm takes only $O(b^3)$ time and $O(b)$ space for a b-bit number.

3.1.2 Discrete Logarithm Problem

Given a generator $\alpha \in G$ of a cyclic group of order n, and given some $\beta = \alpha^x \in G$, the discrete logarithm of β in a base α is the unique value of $x \mod n$. Given β, the problem of finding x is called the Discrete Logarithm Problem.

3.1.3 Quadratic Residuosity Problem

An integer a is said to be quadratic residue modulo n if there exists $0 < x < n$ such that
$x^2 \equiv a \mod n$.
Otherwise, a is said to be a non-quadratic residue modulo n.
If n is an odd prime number, then determining whether or not an integer a is a quadratic residue modulo p is equivalent to calculating Legendre symbol, $(\frac{a}{p})$.
Let the set QR_n denote the set of quadratic residues modulo n. Then given an odd composite integer $n > 3$ and an integer a such that $(\frac{a}{n}) = 1$, the problem of determining if $a \in QR_n$ is called the Quadratic Residue Problem.

3.2 Homomorphic Cryptosystems

3.2.1 RSA Cryptosystem

RSA [18] is a popular public-key cryptosystem both in theory and practice that is used mostly for digital signature. Its strength lies on the intractability of the integer factorization problem described in the previous section. The basic RSA scheme including the key generation, encryption and decryption procedure is described below. Make the $Public\_Key$ available to everyone and keep the $Private\_Key$ a secret.

Algorithm 1 RSA_KeyGeneration()

1: Choose two large prime numbers p and q.

2: Compute $n = pq$ and $\phi(n) = (p-1).(q-1)$.

3: Choose an integer e and an integer d, such that e, is coprime to $\phi(n)$ and $1 < e < \phi(n)$.

4: $ed \equiv 1 \mod m$.

5: $Public\_Key = (e, n)$

6: $Private\_Key = d$

Encryption Procedure

To encrypt, the sender encodes the message into a numerical form M. Encryption is carried as follows:

$$EM \equiv M^e \mod n.$$

Decryption Procedure

To decrypt, the receiver uses the following formula first and then decodes the obtained number to get the intended Message,

$$M \equiv EM^d \mod n.$$

Homomorphic Properties:

The unpadded RSA exhibits the property of multiplicative homomorphism as shown below.

Consider two ciphertexts C_1 and C_2, which are encryptions of plaintexts M_1 and M_2.

Then according to the encryption function of RSA, C_1 can be written as follows,

$C_1 = {P_1}^e \mod n$, where (e, n) is the public key.

Similarly C_2 can be expressed as follows,

$C_2 = {P_2}^e \mod n$, where (e, n) is the same public key.

Now multiplying the ciphertexts as follows,

$C_1.C_2 \mod n = {P_1}^e \mod n.{P_2}^e \mod n = P_1.{P_2}^e \mod n$

This represents a valid encryption for $M_1 * M_2$.
However, in real world applications random padding is used in RSA and the homomorphism does not hold. Hence RSA is not widely used for homomorphic applications. But still it is one of the oldest cryptosystem with homomorphic property.

The security of this system rests upon the difficulty in factoring n into p and q. Hence the numbers p and q must be very large at least 512 bits and n must be at least 1024 bits to ensure security. So far 768-bit RSA has been broken and hence higher key sizes are recommended for present and future usage [26].

3.2.2 Paillier Cryptosystem

Introduced in 1999, Paillier cryptosystem [20] is a probabilistic asymmetric algorithm for public key cryptography that is based on the composite residuosity classes problem.
An integer z is said to be an nth residue $\mod n^2$ if there exists $y \in \mathbb{Z}^*_{n^2}$ such that $z \equiv y^n \mod n^2$. The main idea, is that it is hard to determine whether an arbitrary element in $\mathbb{Z}^*_{n^2}$ is an nth residue $\mod n^2$ without the underlying factorization. This is called the decisional composite residuosity assumption (DCRA).
The key generation, encryption, and decryption procedure, along with the homomorphic property of the cryptosystem is presented below.

Algorithm 2 Paillier_KeyGeneration()

1: Choose two large prime numbers p and q randomly such that $gcd(pq, (p-1)(q-1)) = 1$.
2: Compute $n = pq$ and $\lambda = lcm(p-1, q-1)$
3: Select random integer g where $g \in Z^*_{n^2}$
4: $\mu = (L(g^\lambda \mod n^2))^{-1} \mod n$ where $L(u) = \frac{u-1}{n}$
5: PublicKey $= (n, g)$
6: PrivateKey $= (\lambda, \mu)$

Algorithm 3 Paillier_Encryption()

1: Let m be a message to be encrypted and $m \in Z_n$.

2: Select random r where $r \in Z_n$.

3: Compute the ciphertext c as $c = g^m.r^n \mod n^2$.

Algorithm 4 Paillier_Decryption()

1: Compute the message as $m = L(c^\lambda \mod n^2).\mu \mod n$.

Homomorphic Properties:

Paillier cryptosystem supports the property of additive homomorphism. In this cryptosystem the product of two ciphertexts will decrypt to the sum of their corresponding plaintexts. If m_1 and m_2 are the message to be encrypted, $E()$ and $D()$ are the encryption and decryption function respectively and n is from the $publicKey$, then the additive homomorphism property can be expressed as follows.

$$D(E(m_1, publicKey) * E(m_2, publicKey) \mod n^2) = m_1 + m_2 \mod n$$

Consider c_1 and c_2 are the ciphertexts of the plaintexts m_1 and m_2.
Now following the encryption function of Paillier Cryptosystem,
$c_1 = g^{m_1}.r_1^n \mod n^2$, for some random $r_1 \in Z_n^*$ and $c_2 = g^{m_2}.r_2^n \mod n^2$, for some random $r_2 \in Z_n^*$.
Now $c_1.c_2 = g^{m_1+m_2}.(r_1r_2)^n \mod n^2$.
This represents a valid encryption for $m_1 + m_2$.

3.2.3 Boneh-Goh-Nissim Cryptosystem

Boneh, Goh, and Nissim described a homomorphic public key Cryptosystem [21] based on the subgroup decision problem. In simple terms the subgroup decision problem can be described as follows. Let G be a group of composite order $n = q_1q_2$, then given $x \in G$, the infeasibility in determining whether x belongs to a subgroup of order q_1, is called the subgroup decision problem. Next the key generation, encryption and decryption procedure of the cryptosystem is described.

Algorithm 5 BonehGohNissim_KeyGeneration(τ)

1: Given a security parameter $\tau \in Z^+$, select two random τ-bit primes p and q and set $n = pq \in Z$.

2: Generate a bilinear group G of order n.

3: Let g be a generator of G and $e : GXG \rightarrow G_1$ be the bilinear map.

4: Choose two random generators $g, u \xleftarrow{R} G$ and set $h = u^q$.

5: PublicKey = (n, G, G_1, e, g, h)

6: PrivateKey = p

Algorithm 6 BonehGohNissim_Encryption()

1: Message space consists of integers in the set $\{0, 1, 2, ..., T\}$ with $T < q$.

2: Choose a random $r \xleftarrow{R} \{0, 1, 2, ..., n-1\}$.

3: To encrypt a message m, compute $C = g^m h^r \in G$.

4: Return C as the ciphertext.

Algorithm 7 BonehGohNissim_Decryption()

1: Given a ciphertext C, and the private key p the decryption can done on the basis of the following observation.

2: $C^p = (g^m h^r)^p = (g^p)^m$.

3: Compute m as the discrete log of C^p base $\hat{g}$, where $\hat{g} = g^p$.

4: Return m as the decrypted message.

Homomorphic Properties

This cryptosystem supports additive homomorphism and allows one homomorphic multiplication as well, as demonstrated below.

Let (n, G, G_1, e, g, h) be the public key used for encrypting messages $m_1, m_2 \in \{0, 1, ..., T\}$ to get the ciphertexts $C_1, C_2 \in G_1$. The additive homomorphism can be achieved by computing the product $C = C_1 C_2 h^r$ for a random $r \in \{0, 1, ..., n-1\}$.

Two ciphertexts can also be multiplied once by using bilinear map. If $g_1 = e(g, g)$ and $h_1 = e(g, h)$, then g_1 is of order n whereas h_1 is of order p. For some unknown $\alpha \in Z$, determine $h = g^{\alpha q}$.

Let $C_1 = g^{m_1} h^{r_1} \in G$ and $C_2 = g^{m_2} h^{r_2} \in G$.

Now given $C_1 and C_2$, a valid encryption of the product $m_1.m_2 \mod n$ can be calculated as follows.

Select a random $r \in Z_n$ and calculate $C = e(C_1, C_2) h_1^r \in G_1$.

Now $C = e(C_1, C_2) h_1^r = e(g^{m_1} h^{r_1}, g^{m_2} h^{r_2}) h_1^r$

$= g_1^{m_1 m_2} h_1^{m_1 r_2 + r_2 m_1 + \alpha q r_1 r_2 + r} = g_1^{m_1 m_2} h_1^{\ddot{r}} \in G_1$.

where $\ddot{r} = m_1 r_2 + r_2 m_1 + \alpha q r_1 r_2 + r$ is distributed uniformly in Z_n. Hence C is valid encryption of $m_1 m_2 \mod n \in G_1$.

In terms of application, [27] describes an application of homomorphic encryption for secure aggregation in Sensor Networks. CryptDB [28] is another example that allows SQL queries over encrypted data and uses Paillier Cryptosystem to perform database operations.

Chapter 4

Secure data storage using Homomorphic Encryption

Current Scenario

Proposed Framework

4.1 Current Scenario

The necessity of data storage over the cloud and the challenges associated with protection of outsourced data has been discussed. Common solutions taken by organizations today includes server-side encryption, client-side encryption or the use of both along with strong authentication and authorization schemes to ensure controlled access to the data stored on cloud. But despite of all these measures, security breaches have occurred in past taking advantages of loop holes in the system. Hence new methods in context of cloud security are always welcomed. A framework developed as part of the work, for secure data storage over the cloud is described below.

4.2 Proposed Framework

The proposed framework is for a hybrid cloud environment and can be extended to support multi-cloud scenarios. This is designed with a view towards use in client organizations who already have a private cloud setup. Client organizations who operate using third-party cloud infrastructure can also integrate the framework with their infrastructure. However in such a case it is assumed that the organization has complete trust on its infrastructure provider. In both the cases the objective is to secure the data of the organization stored with other untrusted public clouds. A usage scenario of the framework is depicted in the Fig. 4.1.

Before the framework can be used, some initialization steps need to be followed. The first step involves generation of public and private keys. The public cloud services with which interactions are to be done are provided in the next step. The public key file is sent to all the available public clouds specified in the previous step. This completes the initialization steps.

As new providers are added, the public key needs to be sent to the new providers. The private key is kept secret and is never communicated with any third-party. It is only used within the private cloud to decrypt, the encrypted

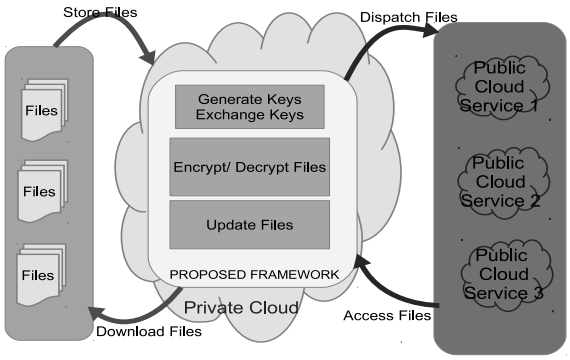

Figure 4.1: Framework Usage Scenario

files. In case the private key is compromised, the initialization steps must be performed again and all files stored using the framework must be encrypted again with the newly generated public key.

The framework mostly helps in storing, distributing, and serving files in a secure manner so that the user do not have to manually deal with the process. The overall flow of data proceeds as follows. First, data generated by various users which could include data from the employees, remote collaborators, employees from third-party organizations on contract, or individuals using services provided by the client organization. This data is stored and uploaded in files to the cloud infrastructure of the organization through an interface. As long as the files remain within the trusted infrastructure of the organization it remains secure, but when it is forwarded to other public cloud services the question on trust arises. The proposed framework offers a convenient method to achieve interaction with other public clouds. The uploaded files are stored in a temporary private storage and are encrypted using Paillier Homomorphic Cryptosystem [20] to ensure confidentiality. Now based on the configuration, the encrypted files are sent to the public clouds. From this point onwards the security of the data is questionable but since the contents are encrypted, confidentiality of the data is ensured. Now at a later stage when access to a file is requested and if the file is available within the private storage then it is returned as response otherwise the file is downloaded from the third-party cloud service on which it is stored and

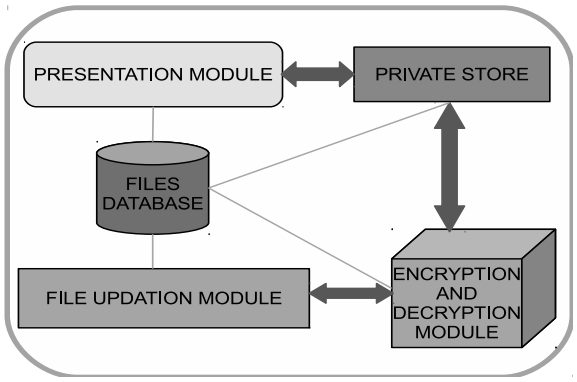

Figure 4.2: Overview of Framework Components

then decrypted within the trusted private storage and the decrypted file is served as response.

4.2.1 Components

The proposed framework consists of five major components as shown in Fig. 4.2. A brief description of each component is described below.

Encryption/Decryption Module

This module is responsible for the key generation, encryption, and decryption of files. It uses the Paillier Cryptosystem [20] for different cryptographic operations. The key generation procedure is the same as described by the Paillier Cryptosystem. The procedure for encrypting and decrypting files are described next. Here the encrypt and decrypt procedures refer to the encryption and decryption methods as described by the Paillier Cryptosystem [20]. To encryption and decryption of files are performed in blocks of fixed size e.g. 128 bytes. The content of each block is converted to a numerical form using their hexadecimal representation. Then the encryption and decryption is performed on the block and the result is appended to the encrypted file. For simplicity the procedure is described below in form of an algorithm.

Algorithm 8 ENCRYPTFILE ($FILE, EFILE, publicKey$)

1: Read $FILE$ till end, in fixed size blocks
2: **for all** blocks **do**
3: $BlockContent \leftarrow$ Content of the block from $FILE$
4: $e \leftarrow$ encrypt($BlockContent$, $publicKey$)
5: Write e to $EFILE$
6: **end for**

Algorithm 9 DECRYPTFILE ($EFILE, DFILE, privateKey$)

1: Read $EFILE$ till end, in fixed size blocks
2: **for all** blocks **do**
3: $BlockContent \leftarrow$ Content of the block from $EFILE$
4: $d \leftarrow$ decrypt($BlockContent$, $privateKey$)
5: Write d to $DFILE$
6: **end for**

Private Store

This framework is designed with a view to forward files to other cloud services. But in order to process the files before sending, a temporary storage area is necessary and Private Store offers that storage. Various operations supported by the private store are storing user files, serving file download requests and sending files to public clouds. Details of the operations are described in STOREFILE, DISPATCHFILE, and ACCESSFILE procedures.

FILES database

FILES database contains basic information on each and every file stored using this framework including the files created afterwards such as encrypted files and patch files. It also includes the list of cloud services where the files need to be stored. Various information stored in the database is given in Fig.4.3. File ID is a unique value assigned to identify the file. The file name and file size attribute contain the name of the file as uploaded by the user and its size respectively. The location

Algorithm 10 STOREFILE ($FILE$)

```
Write FILE to Private Store
Update FILES database
ENCRYPTFILE(FILE)
Search for existing versions of FILE
if found then
    Select the latest existing file version, Fold and remove others, if any
    if size(FILE) > size(Fold) then
        EncryptedDifferencing(Fold, FILE, EFdiff, publicKey)
    end if
end if
DISPATCHFILE(FILE)
```

Algorithm 11 DISPATCHFILE($FILE$)

```
Search for encrypted version of the file, EFILE and encrypted patch file, EFdiff
if EFdiff is found then
    F ← EFdiff
else
    F ← EFILE
    Get the list of public cloud services for file storage, LocationList from the FILES database.
    for cloudService ∈ LocationList do
        Send (F, cloudService)
    end for
    [Optional] Remove EFdiff (if any) and EFILE from Private Store.
end if
```

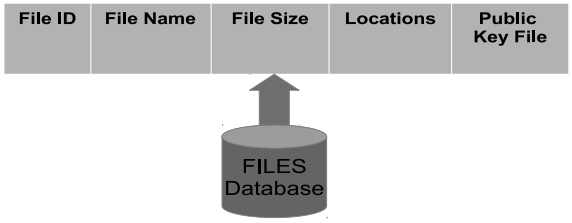

Figure 4.3: File information on FILES database.

attribute contains a list of cloud storage services on which the file is stored. The Public Key File attribute keeps the name of the public key file to be used for encrypting the file. In case same public key file is used for all plaintext files, this attribute will have same value. This attribute is kept for future use. It can be used to encrypt a file with different keys depending on the cloud storage location or other file attributes.

Presentation Module

This is the primary interface for user interaction with the cloud. It is a web interface which provides users a list of their files stored with our framework and allows them to upload or download files. For an administrator, it provides options to change the configuration such as generating new keys, and adding support for new cloud services.

4.2.2 Updation Methods

Now one problem is noticed with this framework. After encryption, the file size becomes huge and consumes high bandwidth during transmission. Further many files get frequent updates in which case with the normal approach, transmission of the encrypted version of the updated file is necessary. To reduce the bandwidth consumption the following approaches can be considered.

File Updation Approach 1: The different versions of the plaintext file can be compared to find the differences among them using tools such as bsdiff [29], or xdelta [30]; and then the generated difference file can be encrypted and sent to

Algorithm 12 ACCESSFILE($PlainFile$)

Query the FILES database for $PlainFile$
if not found **then**
    Print 'File Not Found!'
else
    Scan for the latest version of plaintext file, $PlainFile$ in Private Store.
    if available **then**
        return $PlainFile$ in response
    else
        Search for the encrypted version of $PlainFile$, $EFile$.
        if $EFile$ does not exist in Private Store **then**
            Get the list of public cloud services for file storage, $LocationList$ from the FILES database.
            Download it from any one of the public clouds in $LocationList$
        else
            Use the existing $EFile$
        end if
        DECRYPTFILE($EFile, DFile, privateKey$)
    end if
    return $DFile$ in response
end if

the cloud providers where the file is stored. But as the providers now have the encrypted version of original file, the public key, and encrypted version of the patch file hence to carry out the updation would require the private key file to decrypt the contents first. Therefore for security reasons this approach cannot be adapted.

File Updation Approach 2: As an alternative, the updated version of the plaintext file is encrypted and comparison is made between the respective encrypted versions of the file and then the resulting patch file is sent to the public clouds as it is. As the patch file contains the difference between two encrypted files, its content does not reveal anything of the either plaintext file. Now using tools, the patch file can be used to create an encrypted version of the updated file from the encrypted version of the original plaintext file.

File Updation Approach 3(Recommended approach): Now a modified approach for updating encrypted files that is useful in cloud scenarios is described. In this approach, given two versions of a plaintext file, say F_1 and F_2 which are states of the file at time t_1 and t_2, the objective is to find an encrypted patch file $EFdiff$ that can be used to create an encrypted version of F_2 say EF_2 from the encrypted version of F_1 say EF_1. Now $EFdiff$ can be sent to the public clouds where EF_1 is stored and can be used to update the encrypted file without the need for transmission of the encrypted version of the updated file. The method for finding encrypted difference and performing update using the difference is described in the EncryptedDifferencing and EncryptedPatching procedures. Here the *encryptAdd* procedure uses the additive homomorphism property of the Paillier Cryptosystem to add the encrypted values. The size of the resulting patch file can be further reduced using compression techniques.

Security Analysis:

Consider $M1$ and $M2$ as two instances of a message.

Hence, $M1 = m1_1.m1_2.m1_3...m1_n$

and $M2 = m2_1.m2_2.m2_3...m2_n$, where m_i represents a single block of a message.

Let $C1$ and $C2$ be the ciphertexts generated by applying the encryption procedure on $M1$ and $M2$, respectively.

Algorithm 13 EncryptedDifferencing $(F_1, F_2, EFdiff, publicKey)$

Read F_1 and F_2 till the end, in fixed size blocks
for all blocks **do**
    $BN_1 \leftarrow$ Content of the block from F_1
    $BN_2 \leftarrow$ Content of the block from F_2
    if $BN_1 \neq BN_2$ **then**
        if $BN_2 > BN_1$ **then**
            $d \leftarrow BN_2 - BN_1$
            $findex \leftarrow BlockIndex$
        else
            $d \leftarrow BN_2$
            $findex \leftarrow -(BlockIndex)$
        end if
        $e \leftarrow$ encrypt(d, $publicKey$)
        write ($findex$, e) to $EFdiff$
    end if
end for

Algorithm 14 EncryptedPatching $(EF_1, EFdiff, EF_2, publicKey)$

Read $EFdiff$ till end, in pairs $(findex, difference)$
Read EF_1 and write unaffected blocks to EF_2
for all pairs **do**
    if $findex < 0$ **then**
        $u \leftarrow difference$
    else
        Assign e the encrypted value at $findex$ of EF_1
        $u \leftarrow$ encryptAdd$(e, difference, publicKey)$
    end if
    Write u to EF_2
end for

Now $C1 = c1_1.c1_2.c1_3...c1_n$

and $C2 = c2_1.c2_2.c2_3...c2_n$, where c_i represents a single block of a ciphertext C.

As $M2$ is an updated version of $M1$, hence it can also be expressed as follows

$M2 = (m1_1 + d_1).(m1_2 + d_2).(m1_3 + d_3)...(m1_n + d_n)$,

where $d_i = m2_i - m1_i$, is the difference.

Now a single block of ciphertext can be expressed as,

$c1_i = encrypt(m_i, publicKey)$.

Further the encrypted difference for a block i can be expressed as,

$ed_i = encrypt(d_i, publicKey)$.

As the Paillier Cryptosystem is used for encryption which exhibits the additive homomorphism property,

Hence $decrypt((c1_i * ed_i) \mod n^2, privateKey) = m1_i + d_i = m2_i$

Similarly when this is applied over all the blocks, one can update the message $M1$ to $M2$, by using the ciphertext $C1$ and the encrypted difference.

It is important to note that in the proposed framework the cloud providers do not have the privateKey and hence cannot decrypt the ciphertexts. However they can update the content in their encrypted form itself. All interactions with the untrusted cloud involves encrypted contents only.

Chapter 5

Implementation and Results

Experiment Details

Performance Analysis

A prototype to evaluate the working of the proposed framework has been developed in Python. The Paillier homomorphic cryptosystem and associated cryptographic operations are implemented using Sage [31]. The framework has been tested only in local environment as described in section 5.1. This offers details on the performance of the proposed framework with respect to the performance parameter i.e. the file size. For a full-scale cloud implementation a python server process running in the background is needed to keep the framework alive and to provide the client interface.

5.1 Experiment Details

To evaluate the framework eight different text files were considered. Each of these files were updated 7 times which resulted into a new state for a file after each update. Considering the state or version of all eight files after a change as an instance, there were total of eight instances including the state of the original text files. For each instance every file is encrypted, and the patch file if possible is generated with respect to the previous instance. As described earlier the patch file is generated using the proposed updation approach and using two popular tools xdelta [30], and bsdiff [29]. The size of the encrypted file, the patch file generated using xdelta, bsdiff, and proposed approach for three instances for all the 8 text files are tabulated in Table 5.1. After generation the patch files using the proposed approach were compressed for further reduction in size.

5.2 Performance Analysis

In order to understand the performance the different approaches, a plot of change in file sizes for the encrypted and various patch files over the eight instances is presented in figure 5.1. For a given instance the average value of the eight file sizes are considered for each type. It has been observed that the proposed

Table 5.1: Various file size information for 3 instances.

Files	Encrypted File Size in KB	Patch File Size in KB		
		xdelta	bsdiff	Our Approach
File 1	1072	616	464	412
	2968	1700	1288	188
	5272	3020	2288	716
File 2	620	356	268	200
	1428	820	620	140
	2536	1452	1100	296
File 3	1036	596	448	388
	2708	1552	1176	156
	4768	2732	2072	556
File 4	1280	732	556	516
	3800	2176	1652	48
	6600	3784	2872	616
File 5	848	488	368	308
	2260	1296	980	132
	3812	2184	1656	396
File 6	448	260	196	100
	848	484	368	36
	1412	812	612	144
File 7	1172	672	508	480
	3172	1816	1376	256
	5596	3204	2432	616
File 8	2348	1344	1016	928
	7112	4076	3096	540
	10640	6100	4640	932

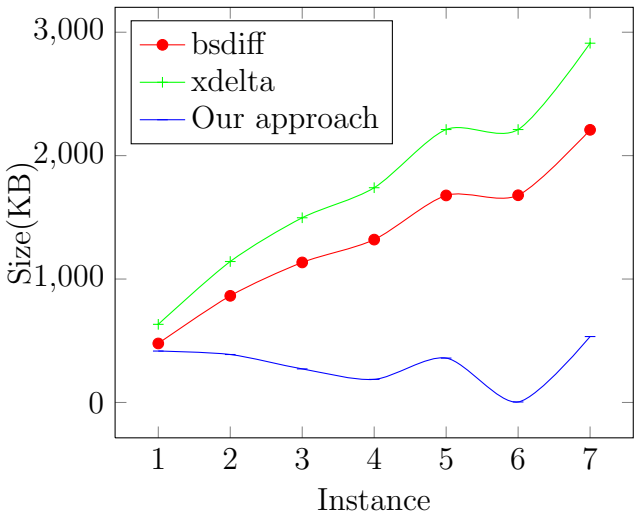

Figure 5.1: Variation of patch file sizes over instances

approach generates smaller patch files on an average. It is also observed that patch files generated using the proposed approach are small compared to the ones produced by other tools in their current state. This is due to the fact that other tools were not designed with security applications in mind and hence have to compare and produce the patch based on the encrypted version of the files, whereas our approach compares the plaintext versions of the files and then encrypts the differences and uses the property of additive homomorphism to apply the patches.

Another way to visualize the performance is to analyze the reduction in size of the file to be transmitted. For this a parameter α which is defined as follows is used.

$$\alpha = \frac{PatchFileSize}{EncryptedFilesize} * 100\% \tag{5.1}$$

A plot of α for the files considered over the different instances is presented in figure 5.2.

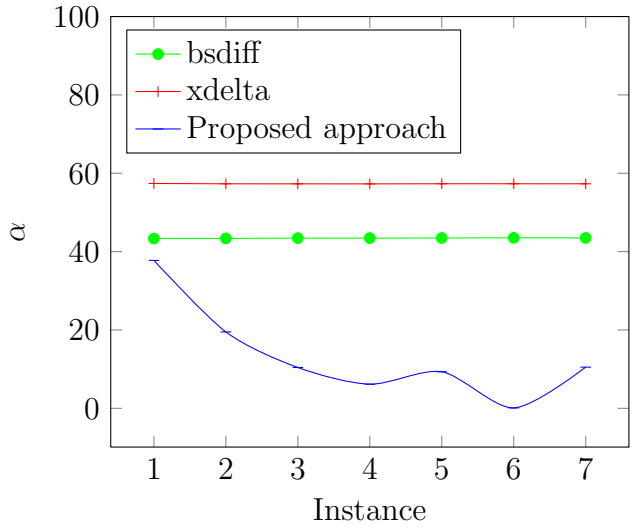

Figure 5.2: Variation of α over instances

Chapter 6

Conclusion and Future work

Conclusion

Suggestions for Improvement

To summarize, the work gives a model of a framework that can be used by organizations to protect and manage their data stored over untrusted public clouds. As part of the work the possibility of using delta encoding concepts along with homomorphic encryption scheme with additive homomorphism to update encrypted files, instead of transmitting entire encrypted versions each time after an update, was explored. Under the test environment, the developed prototype has delivered promising performance results as compared to other common solutions. Hence the proposed approach might be considered for use in real world scenarios.

Suggestions for Improvement

The included support for the choice of cloud providers is limited and very basic, but the framework can be extended to support new providers. Further in the current implementation the organizations have to maintain an application at the public clouds that will perform the updation procedure. The framework can be developed in a manner such that the providers can easily integrate it with their platform. Moreover working with encrypted data is computation intensive and expensive in terms of storage, hence high performance data processing options in the cloud can be applied for better performance. With reference to cryptographic techniques, the proposed approach using newer homomorphic cryptosystems having additive homomorphism, can be explored for performance benefits.

Bibliography

[1] Ponemon research study infographic: Whos minding your cloud? http://www.ca.com/us/collateral/white-papers/na/ponemon-research-study-infographic-whos-minding-your-cloud.aspx, 2013.

[2] Dropbox. https://www.dropbox.com/.

[3] Dropbox confirms it was hacked, offers users help. http://news.cnet.com/8301-1009_3-57483998-83/dropbox-confirms-it-was-hacked-offers-users-help/.

[4] Last.fm. http://www.last.fm/.

[5] Last.fm password security update. http://www.last.fm/passwordsecurity, 2012.

[6] icloud. https://www.icloud.com/.

[7] Another apple disaster: The icloud gets hacked. http://www.forbes.com/sites/timworstall/2012/08/07/another-apple-disaster-the-icloud-gets-hacked/.

[8] Securing the clouds [infographic]. http://www.tappin.com/blog/2012/12/cloud-security-infographic/, 2012.

[9] Peter Mell and Tim Grance. The NIST definition of cloud computing. Technical report, July 2009.

[10] Deyan Chen and Hong Zhao. Data security and privacy protection issues in cloud computing. In *Computer Science and Electronics Engineering (ICCSEE), 2012 International Conference on*, volume 1, pages 647–651, 2012.

[11] M.A. AlZain, E. Pardede, B. Soh, and J.A. Thom. Cloud computing security: From single to multi-clouds. In *System Science (HICSS), 2012 45th Hawaii International Conference on*, pages 5490–5499, 2012.

[12] Steven Y. Ko, Kyungho Jeon, and Ramsés Morales. The hybrex model for confidentiality and privacy in cloud computing. In *Proceedings of the 3rd USENIX conference on Hot topics in cloud computing*, HotCloud'11, pages 8–8, Berkeley, CA, USA, 2011. USENIX Association.

[13] Witold Litwin, Sushil Jajodia, and Thomas Schwarz. Privacy of data outsourced to a cloud for selected readers through client-side encryption. In *Proceedings of the 10th annual ACM workshop on Privacy in the electronic society*, WPES '11, pages 171–176, New York, NY, USA, 2011. ACM.

[14] A. Patrascu, D. Maimut, and E. Simion. New directions in cloud computing. a security perspective. In *Communications (COMM), 2012 9th International Conference on*, pages 289–292, 2012.

[15] Wayne Jansen and Timothy Grance. Sp 800-144. guidelines on security and privacy in public cloud computing. Technical report, Gaithersburg, MD, United States, 2011.

[16] S. Subashini and V. Kavitha. Review: A survey on security issues in service delivery models of cloud computing. *J. Netw. Comput. Appl.*, 34(1):1–11, January 2011.

[17] R. Rivest, L. Adleman, and M. Dertouzos. On data banks and privacy homomorphisms. pages 169–177. Academic Press, 1978.

[18] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, February 1978.

[19] Taher El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *Proceedings of CRYPTO 84 on Advances in cryptology*, pages 10–18, New York, NY, USA, 1985. Springer-Verlag New York, Inc.

[20] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Proceedings of the 17th international conference on Theory and application of cryptographic techniques*, EUROCRYPT'99, pages 223–238, Berlin, Heidelberg, 1999. Springer-Verlag.

[21] Dan Boneh, Eu-Jin Goh, and Kobbi Nissim. Evaluating 2-dnf formulas on ciphertexts. In *Proceedings of the Second international conference on Theory of Cryptography*, TCC'05, pages 325–341, Berlin, Heidelberg, 2005. Springer-Verlag.

[22] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the 41st annual ACM symposium on Theory of computing*, STOC '09, pages 169–178, New York, NY, USA, 2009. ACM.

[23] Zvika Brakerski and Vinod Vaikuntanathan. Fully homomorphic encryption from ring-lwe and security for key dependent messages. In *Proceedings of the 31st annual conference on Advances in cryptology*, CRYPTO'11, pages 505–524, Berlin, Heidelberg, 2011. Springer-Verlag.

[24] Jean-Sébastien Coron, David Naccache, and Mehdi Tibouchi. Public key compression and modulus switching for fully homomorphic encryption over the integers. In *Proceedings of the 31st Annual international conference on Theory and Applications of Cryptographic Techniques*, EUROCRYPT'12, pages 446–464, Berlin, Heidelberg, 2012. Springer-Verlag.

[25] Craig Gentry, Shai Halevi, and Nigel P. Smart. Fully homomorphic encryption with polylog overhead. In *Proceedings of the 31st Annual international conference on Theory and Applications of Cryptographic Techniques*, EUROCRYPT'12, pages 465–482, Berlin, Heidelberg, 2012. Springer-Verlag.

[26] Thorsten Kleinjung, Kazumaro Aoki, Jens Franke, Arjen K. Lenstra, Emmanuel Thomé, Joppe W. Bos, Pierrick Gaudry, Alexander Kruppa, Peter L. Montgomery, Dag Arne Osvik, Herman Te Riele, Andrey Timofeev, and Paul Zimmermann. Factorization of a 768-bit rsa modulus. In *Proceedings of the 30th annual conference on Advances in cryptology*, CRYPTO'10, pages 333–350, Berlin, Heidelberg, 2010. Springer-Verlag.

[27] R. Riggio and S. Sicari. Secure aggregation in hybrid mesh/sensor networks. In *Ultra Modern Telecommunications Workshops, 2009. ICUMT '09. International Conference on*, pages 1–6, 2009.

[28] Raluca Ada Popa, Catherine M. S. Redfield, Nickolai Zeldovich, and Hari Balakrishnan. Cryptdb: protecting confidentiality with encrypted query processing. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, SOSP '11, pages 85–100, New York, NY, USA, 2011. ACM.

[29] Colin Percival. Naive differences of executable code, 2003.

[30] Joshua P. MacDonald. File system support for delta compression. Technical report, 2000.

[31] Sage: Open source mathematics software. `http://www.sagemath.org/`.